Compréhension et utilisation de 3 packages de R: evir, evd, et graphics

Thuy AUFRERE, Nina ZOUMANIGUI, Arnaud Bruel YANKO

08/12/2020

1. Introduction

Dans une session R, nous avons accès à un bon nombre de fonctions et de jeux de données. Les objets accessibles sont ceux contenus dans les packages R chargés à l'ouverture de la session. Un package R est simplement un regroupement de fonctions et de données documentées. Ce tutoriel à pour but de vous faire une présentation de certains packages qui ont été choisis à savoir evir, evd, R.miner, graphics, packages qui nous permettrons:

- D'explorer le monde de la théorie des valeurs extrêmes Cours intéressant sur la Statisque des valeurs extrêmes en modélisant les risques extrêmes;
- Travailler d'une manière générale sur la régression.

2. Installation et chargement d'un package R

L'installation d'un package et le chargement d'un package sont deux étapes distinctes. Certains packages *R* sont installés automatiquement lors de l'installation de *R*.

La fonction installed.packages retourne des informations à propos des packages *R* installés sur l'ordinateur local. Il est simple de charger en *R* des packages supplémentaires à ceux chargés par défaut. Il suffit d'utiliser les commandes comme dans l'exemple suivant :

```
installed.packages
```

```
library(evir)
```

```
Arnaud<-installed.packages()
head(Arnaud, n=3)
```

```
##        Package  LibPath                                        Version Priority
## abind  "abind"  "C:/Users/yanko/Documents/R/win-library/3.6" "1.4-5" NA
## actuar "actuar" "C:/Users/yanko/Documents/R/win-library/3.6" "2.3-3" NA
## adabag "adabag" "C:/Users/yanko/Documents/R/win-library/3.6" "4.2"   NA
##        Depends                              Imports                   LinkingTo
## abind  "R (>= 1.5.0)"                       "methods, utils"          NA
## actuar "R (>= 3.3.0)"                       "stats, graphics, expint" "expint"
## adabag "rpart, caret, foreach, doParallel" NA                        NA
##        Suggests  Enhances License       License_is_FOSS License_restricts_use
## abind  NA        NA       "LGPL (>= 2)" NA              NA
## actuar "MASS"    NA       "GPL (>= 2)"  NA              NA
## adabag "mlbench" NA       "GPL (>= 2)"  NA              NA
##        OS_type MD5sum NeedsCompilation Built
## abind  NA      NA     "no"             "3.6.0"
## actuar NA      NA     "yes"            "3.6.2"
## adabag NA      NA     "no"             "3.6.3"
```

2.1 Evir et evd

Evir et Evd : Qu'est ce que c'est?

Evir est un package de *R* utiliser dans la théorie de valeurs extrêmes, qui peuvent être divisées dans les groupes suivants; analyse exploratoire des données, maxima de bloc, pics au-dessus d'un seuil (univarié et bivarié), processus ponctuels, distributions *gev/gpd*.

Quant-à

Evd il étend les fonctions de simulation, de distribution, de quantile de valeurs extrêmes paramétriques univariées et multivariées, et fournit d'ajustement qui calculent les estimations du maximum de vraisemblance pour des modèles maxima univariées et bivariées, et pour les modèles à seuil univariées et bivariées.

Les fonctions d'Evir et Evd

Ici nous allons recenser certaines fonctions qui peuvent être utilisées dans ces deux packages, commençons par *Evir*.

Listes des fonctions de Evir

Fonctions	Rôles
dgev	Renvoie la distribution des valeurs extrêmes généralisées
dgpd	Distribution de la *Pareto* généralisée
emplot	Graphique de la fonction de distribution empirique

Fonctions	Rôles
findthresh	Permet de trouver le seuil
gev	Permet de trouver le seuil
gpd	Permet d'ajuster les valeurs des valeurs extrêmes généralisées
gumbel	Permet d'ajuste la distribution de *Gumbell*
nidd.annual	Les données de la rivière *Nidd*
pgev	Donne la valeur de la distribution des valeurs extrêmes généralisées
interpret.gpdbiv	Interprétation des résultats de l'ajustement *pgd* bivarié
rgpd	Distribution de la *Pareto* généralisée

Listes des fonctions de Evd

Fonctions	Rôles
failure	Temps d'échec
dextreme	Distributions des maxima et es minima
dmvevd	Distributions paramétriques et valeurs extrêmes multivariées
fextreme	Ajustement du maximum de vraisemblance des maxima et des minima
qfrechet	Permet de trouver le seuil
gpd	Distribution de *Fréchet*
qnweibull	distribution inverse de *weibull*
gumbel	Permet d'ajuste la distribution de *Gumbell*
rorder	Distributions des statistiques d'ordres
confint.evd	Calcule les intervalles de confiance

2.2 Accéder au contenu d'un package R chargé

Une fois un package chargé en R avec la commande `library`, son contenu est accessible dans la session R. Nous avons vu dans des notes précédentes comment fonctionne l'évaluation d'expressions en R. Nous savons donc que le chargement d'un nouveau package ajoute un environnement dans le chemin de recherche de R, juste en dessous de l'environnement de travail. Le chargement de certains packages provoque aussi le chargement de packages dont ils dépendent.

Ainsi, parfois plus d'un environnement est ajouté au chemin de recherche de R lors du chargement d'un package. L'environnement d'un package contient les fonctions publiques et les données du package.

Jeux de données

Souvent, les jeux de données inclus dans un package se retrouvent directement dans l'environnement d'un package dans le chemin de recherche. C'est le cas, par exemple, des jeux de données du package datasets.

```
head(ls("package:datasets"), n = 8)
```

```
head(ls("package:datasets"), n=8)
```

```
## [1] "ability.cov"   "airmiles"      "AirPassengers" "airquality"
## [5] "anscombe"      "attenu"        "attitude"      "austres"
```

Dans notre cas espèce seulement les 8 premiers éléments de la liste sont affichés ici, car cette liste compte normalement 104 éléments.

Cependant, les jeux de données sont parfois cachés. Ils sont alors traités différemment des fonctions privées et ne se retrouvent même pas dans l'espace de noms du package.

La fonction data est très utile dans ce cas. Cette fonction a en fait plusieurs utilités.

Premièrement, elle permet d'énumérer tous les jeux de données contenus dans un package.

```
data(package = "evir")
```

Dans notre cas espèce, nous allons à partir du packages “evir”, afficher le jeu de données *nidd.annual*, ces données représentent les niveaux maximums annuels de la *rivière Nidd* dans le Yorkshire.

On a:

```
library(evd)
```

```
## Warning: package 'evd' was built under R version 3.6.2
```

```
library(evir)
```

```
##
## Attaching package: 'evir'
```

```
## The following objects are masked from 'package:evd':
##
##     dgev, dgpd, pgev, pgpd, qgev, qgpd, rgev, rgpd
```

```
library(actuar)
```

```
## Warning: package 'actuar' was built under R version 3.6.2
```

```
##
## Attaching package: 'actuar'
```

```
## The following objects are masked from 'package:evd':
##
##     dgumbel, pgumbel, qgumbel, rgumbel
```

```
## The following object is masked from 'package:grDevices':
##
##     cm
```

```
library(rmutil)
```

```
## Warning: package 'rmutil' was built under R version 3.6.3
```

```
##
## Attaching package: 'rmutil'
```

```
## The following objects are masked from 'package:actuar':
##
##     dburr, dinvgauss, dpareto, mexp, pburr, pinvgauss, ppareto, qburr,
##     qinvgauss, qpareto, rburr, rinvgauss, rpareto
```

```
## The following object is masked from 'package:stats':
##
##     nobs
```

```
## The following objects are masked from 'package:base':
##
##     as.data.frame, units
```

```
data(nidd.annual)
nidd<-nidd.annual
nidd
```

```
##  [1]  65.08  65.60  75.06  76.22  78.55  81.27  86.93  87.76  88.89  90.28
## [11]  91.80  91.80  92.82  95.47 100.40 111.54 111.74 115.52 131.82 138.72
## [21] 148.63 149.30 151.79 153.04 158.01 162.99 172.92 179.12 181.59 189.04
## [31] 213.70 226.48 251.96 261.82 305.75
```

Nous obtenons en sortie un vecteur numérique contenant 35 observations.

2.3 Applications

Pour application on utilise les données des crues annuelles de la rivière *Nidd*, dans le Yorkshire.

Contexte

La hauteur d'une rivière est modélisée par une variable aléatoire X.

On dispose de $\{X_1, \cdots, X_n\}$ un échantillon de hauteurs d'eau annuelles. On note $X_{1,n} \leq X_{2,n} \leq, \cdots, \leq X_{n,n}$ l'échantillon ordonné, avec $n \in \mathbb{N}$.

Deux problèmes complémentaires:

- Calculer la probabilité p d'une hauteur d'eau h extrême

  $p = \mathbf{P}(X \geq h)$ avec $h > X_{n,n}$.

- Calculer le niveau d'eau h qui est atteint ou dépassé une seule fois sur $T > n$, i.e. résoudre

  $\frac{1}{T} = \mathbf{P}(X \geq h)$

Modélisation : Le but ici étant d'écrire un programme permettant de calculer la fonction empirique des excès moyens et de tracer les différents seuils:

Application 1

- On Charge au préalable nos deux packages, les bouts de code suivant permettent de faire ce travail:

```
library(evd)
```

```
library(evir)
```

- Nous allons commencer par lire les données du fichier *Nidd* et les classer par ordre décroissant dans un vecteur noté *nidd*. On crée un vecteur *seuil* ordonné par ordre.

```
nidd2<-sort(nidd, decreasing = TRUE)
nidd2
```

```
##  [1] 305.75 261.82 251.96 226.48 213.70 189.04 181.59 179.12 172.92 162.99
## [11] 158.01 153.04 151.79 149.30 148.63 138.72 131.82 115.52 111.74 111.54
## [21] 100.40  95.47  92.82  91.80  91.80  90.28  88.89  87.76  86.93  81.27
## [31]  78.55  76.22  75.06  65.60  65.08
```

```
seuil=seq(70,300,by =5)
#seuil<-70:300
seuil
```

```
##  [1]  70  75  80  85  90  95 100 105 110 115 120 125 130 135 140 145 150 155 160
## [20] 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255
## [39] 260 265 270 275 280 285 290 295 300
```

```
taille_seuil=length(seuil)
taille_nidd2=length(nidd2)
```

- Afin de calculer la moyenne des excès pour chaque seuil, on crée une matrice dont le nombre de lignes correspond à la taille du vecteur "*seuil*" et le nombre de colonnes correspond à la taille du vecteur "*nidd*" telle que chaque ligne i correspond aux excès au delà du $seuil[i]$.

Puisque certains excès peuvent être négatifs, nous mettons les zéros à la place des termes négatifs :

```
X=matrix(0,nrow=length(seuil),ncol=length(nidd2))
head(X, n=3)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
## [3,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0
##      [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
## [1,]     0     0     0     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0     0     0     0
## [3,]     0     0     0     0     0     0     0     0     0     0     0     0
##      [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]     0     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0     0
## [3,]     0     0     0     0     0     0     0     0     0
```

```
#View(X)

for(i in 1:length(seuil))
{for(j in 1:length(nidd2))
    {X[i,j]=max(nidd2[j]-seuil[i],0)}
}
head(X, n=3)
```

```
##        [,1]   [,2]   [,3]   [,4]  [,5]   [,6]   [,7]   [,8]   [,9] [,10] [,11]
## [1,] 235.75 191.82 181.96 156.48 143.7 119.04 111.59 109.12 102.92 92.99 88.01
## [2,] 230.75 186.82 176.96 151.48 138.7 114.04 106.59 104.12  97.92 87.99 83.01
## [3,] 225.75 181.82 171.96 146.48 133.7 109.04 101.59  99.12  92.92 82.99 78.01
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23]
## [1,] 83.04 81.79  79.3 78.63 68.72 61.82 45.52 41.74 41.54  30.4 25.47 22.82
## [2,] 78.04 76.79  74.3 73.63 63.72 56.82 40.52 36.74 36.54  25.4 20.47 17.82
## [3,] 73.04 71.79  69.3 68.63 58.72 51.82 35.52 31.74 31.54  20.4 15.47 12.82
##      [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]  21.8  21.8 20.28 18.89 17.76 16.93 11.27  8.55  6.22  5.06     0     0
## [2,]  16.8  16.8 15.28 13.89 12.76 11.93  6.27  3.55  1.22  0.06     0     0
## [3,]  11.8  11.8 10.28  8.89  7.76  6.93  1.27  0.00  0.00  0.00     0     0
```

- On calcule pour chaque seuil la moyenne des excès:

```
somme=rep(0,length(seuil))
somme
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0 0 0 0
```

```
Compteur=rep(0,length(seuil))
Compteur
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [39] 0 0 0 0 0 0 0 0 0
```

```
e=c()
e
```

```
## NULL
```

```
for(i in 1:length(seuil))
{
    for(j in 1:length(nidd2))
    {
        if ( X[i,j]>0 )
```

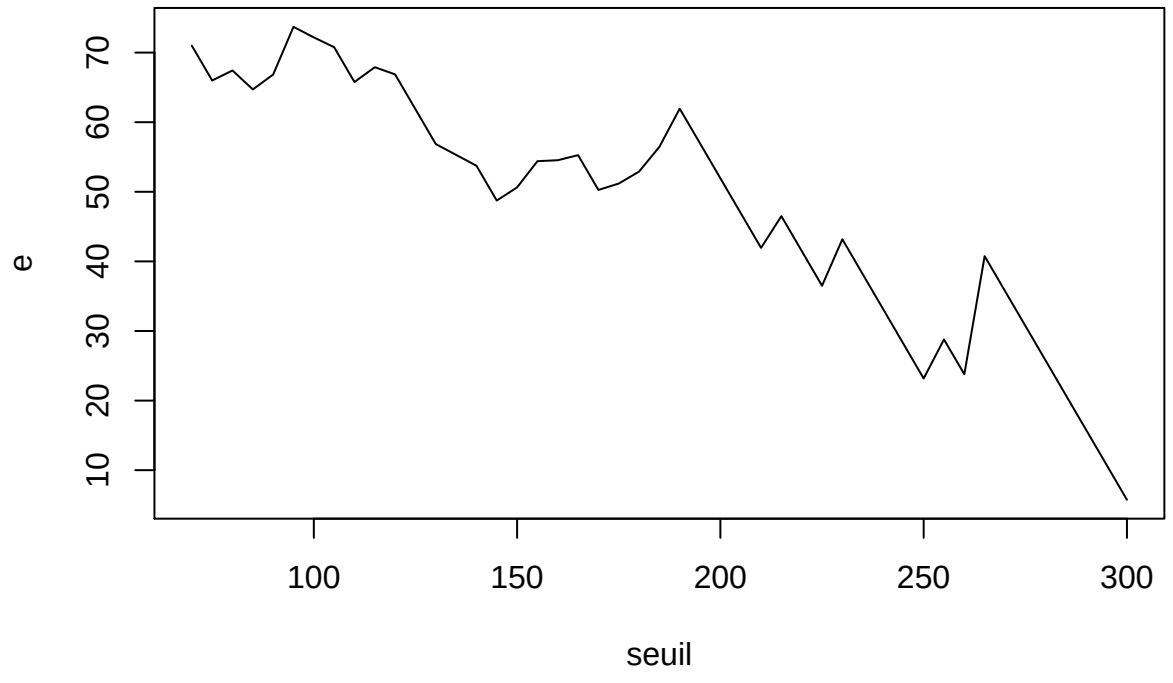
```
        {
            somme[i]=somme[i]+X[i,j]
            Compteur[i]=Compteur[i]+1
        }
    }
    e[i]=somme[i]/Compteur[i]
}
e
```

```
##  [1] 70.99182 65.99182 67.43000 64.71138 66.84808 73.69773 72.18476 70.77400
##  [9] 65.77400 67.90000 66.86353 61.86353 56.86353 55.30375 53.74267 48.74267
## [17] 50.63154 54.39818 54.53700 55.26444 50.26444 51.18250 52.90571 56.45833
## [25] 61.94200 56.94200 51.94200 46.94200 41.94200 46.50250 41.50250 36.50250
## [33] 43.17667 38.17667 33.17667 28.17667 23.17667 28.78500 23.78500 40.75000
## [41] 35.75000 30.75000 25.75000 20.75000 15.75000 10.75000  5.75000
```

- La fonction moyenne des excès est une méthode permettant de répondre à l'une des dificultés de la modélisation des sinistres extrêmes qui est la détermination du seuil. Ainsi, graphiquement il est possible de déterminer le seuil le plus adéquat, en prenant la valeur à partir de laquelle la fonction moyenne des excès est linéaire.

Regardons ensemble ce que cela produit:

```
plot(seuil,e,type='l')
```



Dans notre cas, La fonction moyenne des excès est linéaire à partir de la valeur 260.

- Ici on trace sur deux graphiques juxtaposés:

la fonction empirique des excès moyens en fonction des statistiques d'ordre $x_{(n-k)}$ et en fonction de k.

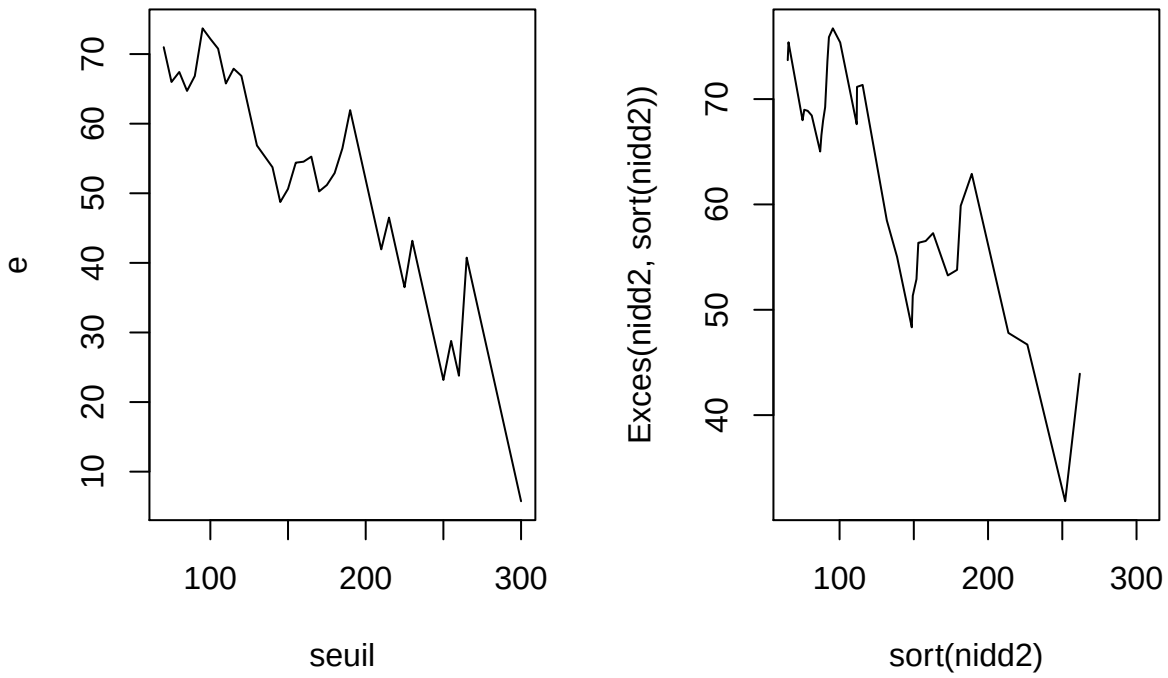
Le $1er$ représente la moyenne des dépassements des crues annuelles de la rivière $Nidd$ par rapport à un certain seuil (indépendant de nos données de départ).

Le $2eme$ représente la moyenne des dépassements de la crue annuelle de la rivière $Nidd$ d'une année par rapport aux autres.

```
Exces=function(observations,seuil)
{
    Exces2=matrix(0,nrow=length(seuil),
    ncol=length(observations))
    Exces2
    #View(X)

    for(i in 1:length(seuil))
    {
        for(j in 1:length(observations))
        {
            Exces2[i,j]=max(observations[j]-seuil[i],0)
        }
    }
    Exces2
    somme2=rep(0,length(seuil))
    somme2
    Compteur2=rep(0,length(seuil))
    Compteur2
    e2=c()
    e2
    for(i in 1:length(seuil))
    {
        for(j in 1:length(observations))
        {
            if ( Exces2[i,j]>0 )
            {
                somme2[i]=somme2[i]+Exces2[i,j]
                Compteur2[i]=Compteur2[i]+1
            }
        }
        e2[i]=somme2[i]/Compteur2[i]
    }
    e2
}


Exces(nidd2,sort(nidd2))
```

```
##  [1] 73.69441 75.39182 67.99219 68.98806 68.88000 68.44138 65.02357 66.57111
##  [9] 67.95808 69.23080 73.59870 73.59870 75.87773 76.71476 75.37400 67.61474
## [17] 71.16000 71.34353 58.48375 55.02267 48.33500 51.33154 52.91167 56.35818
## [25] 56.52700 57.27444 53.26250 53.78571 59.86833 62.90200 47.80250 46.69667
## [33] 31.82500 43.93000      NaN
```

```
par(mfrow=c(1,2))
plot(seuil,e,type='l')
plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
```



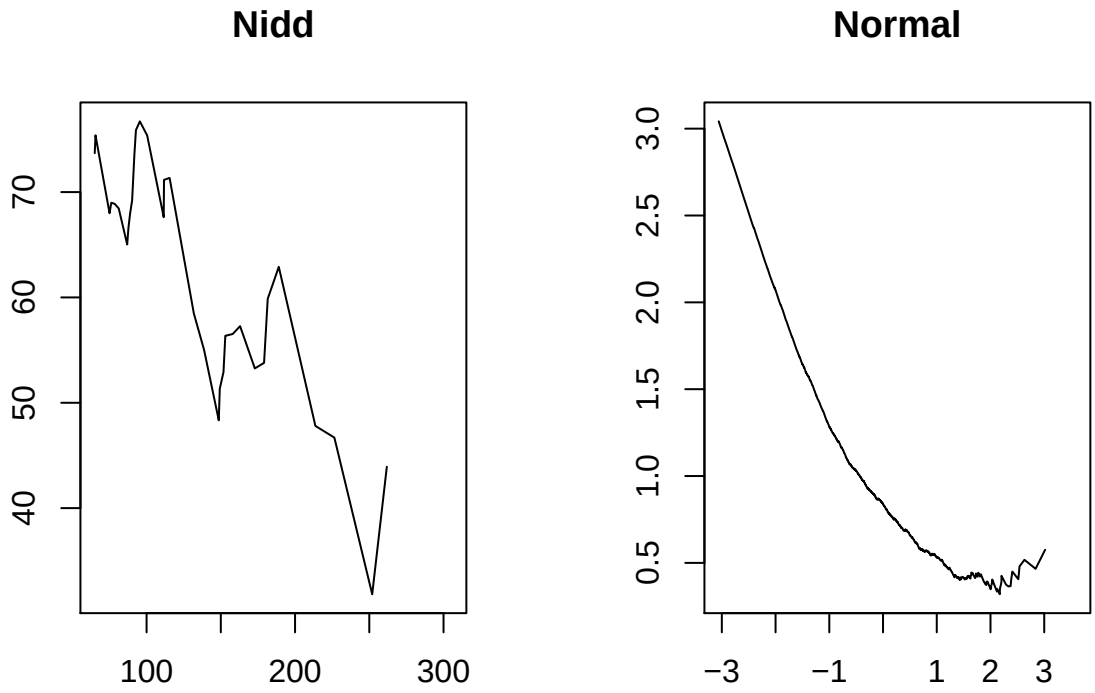
Les deux graphique sont la même allure, et la fonction moyenne des excès devient linéaire aux environs de la valeur 260.

Pour aller plus loin l'idéal serait d'appliquer cela aux différentes lois usuelles: La loi Pareto, Weibull, Frechet, Gamma, Cauchy, Normal et bien d'autres: Les differents lois en Statisque des valeursextrêmes

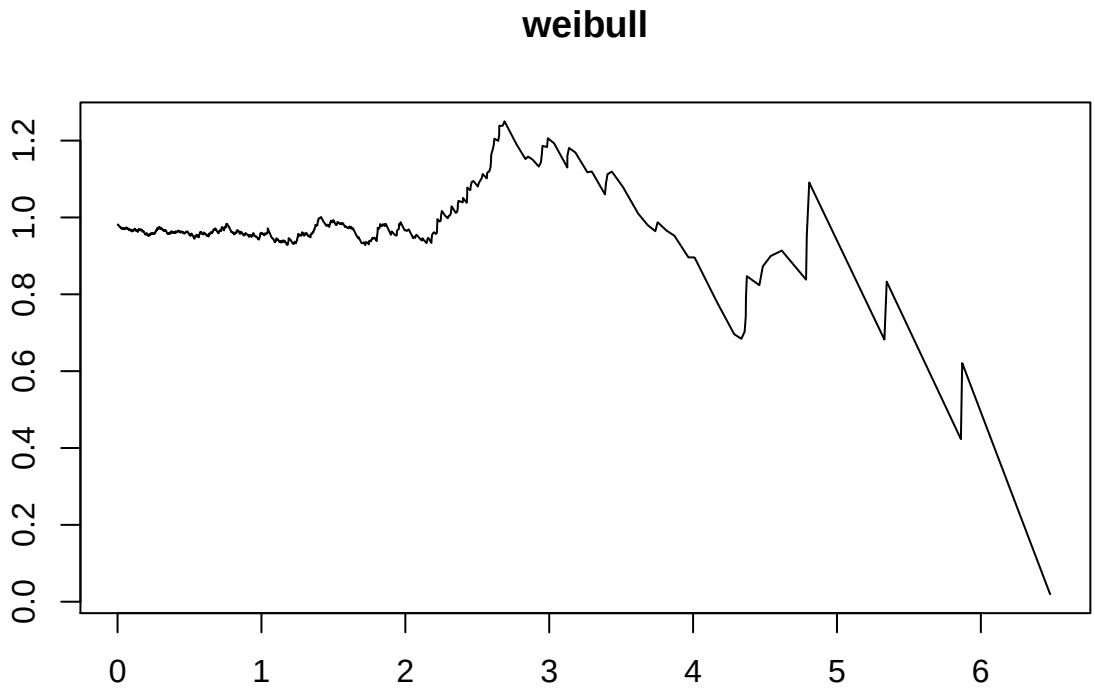
- Application aux lois usuelles

```
###########loi Normal Centrée réduite###########
Nidd_normal=rnorm(1000,0,1)
Nidd_normal_dec=sort(Nidd_normal, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_normal, decreasing = FALSE)
```

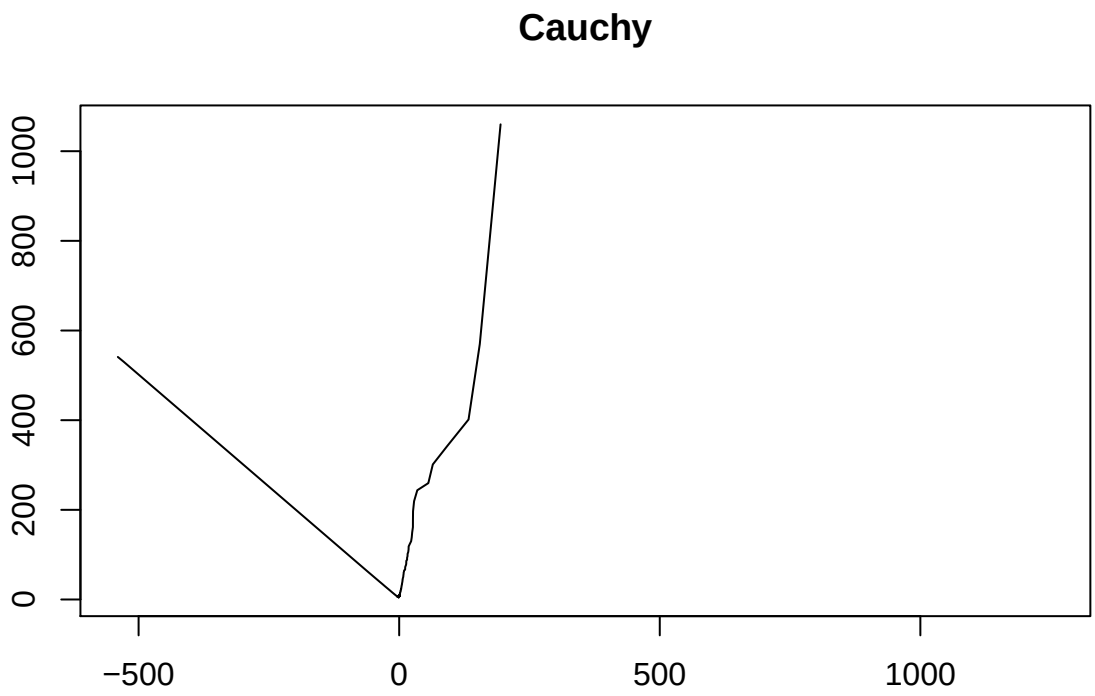
```
#Exces(Nidd_normal_dec,seuil_stat_ord)
par(mfrow=c(1,2))
plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l',main='Nidd',
xlab = "" , ylab = "")
plot(seuil_stat_ord,Exces(Nidd_normal_dec,seuil_stat_ord),type='l',
main='Normal',xlab = "" , ylab = "")
```



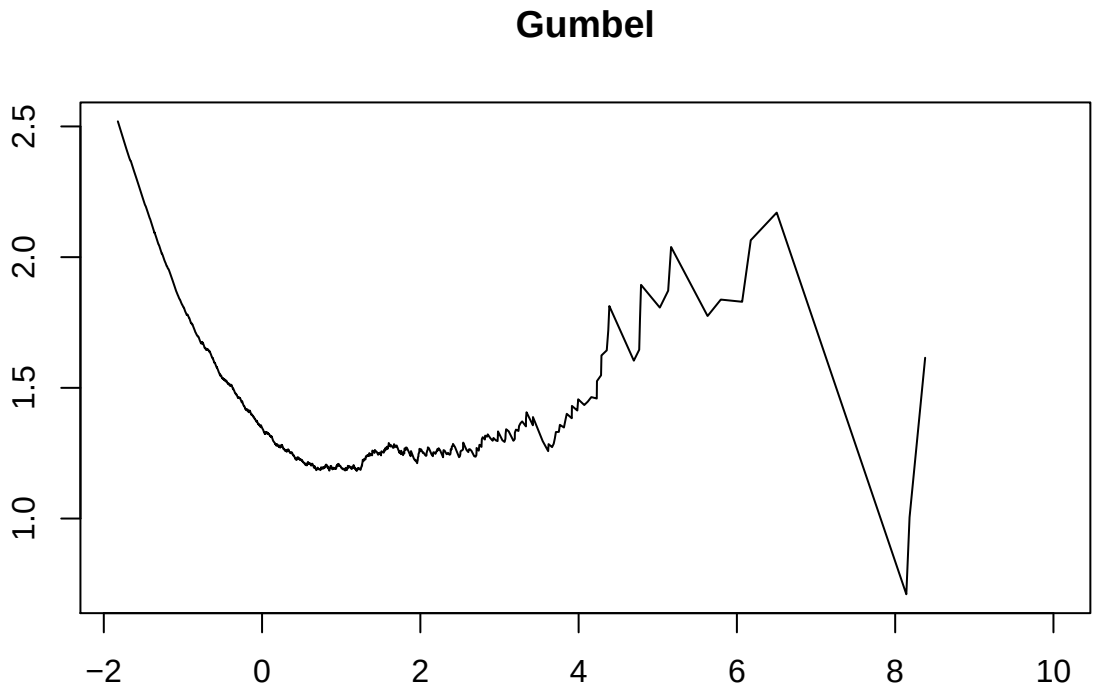

```
############loi Weibull############
Nidd_weibull=rweibull(1000, 1)
Nidd_weibull_dec=sort(Nidd_weibull, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_weibull, decreasing = FALSE)
#Exces(Nidd_weibull_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_weibull_dec,seuil_stat_ord),
type='l',main='weibull',xlab = "" , ylab = "")
```



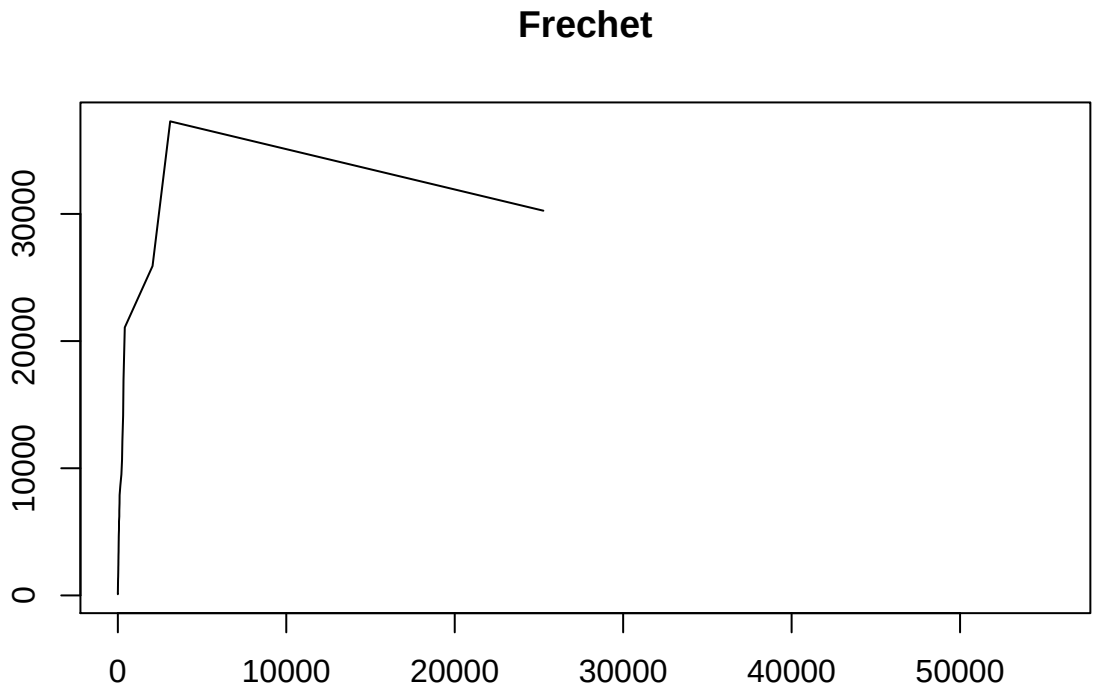

```
############loi Cauchy############
Nidd_cauchy=rcauchy(1000, 0, 1)
Nidd_cauchy_dec=sort(Nidd_cauchy, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_cauchy, decreasing = FALSE)
#Exces(Nidd_cauchy_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_cauchy_dec,seuil_stat_ord),
type='l',main='Cauchy',xlab = "" , ylab = "")
```



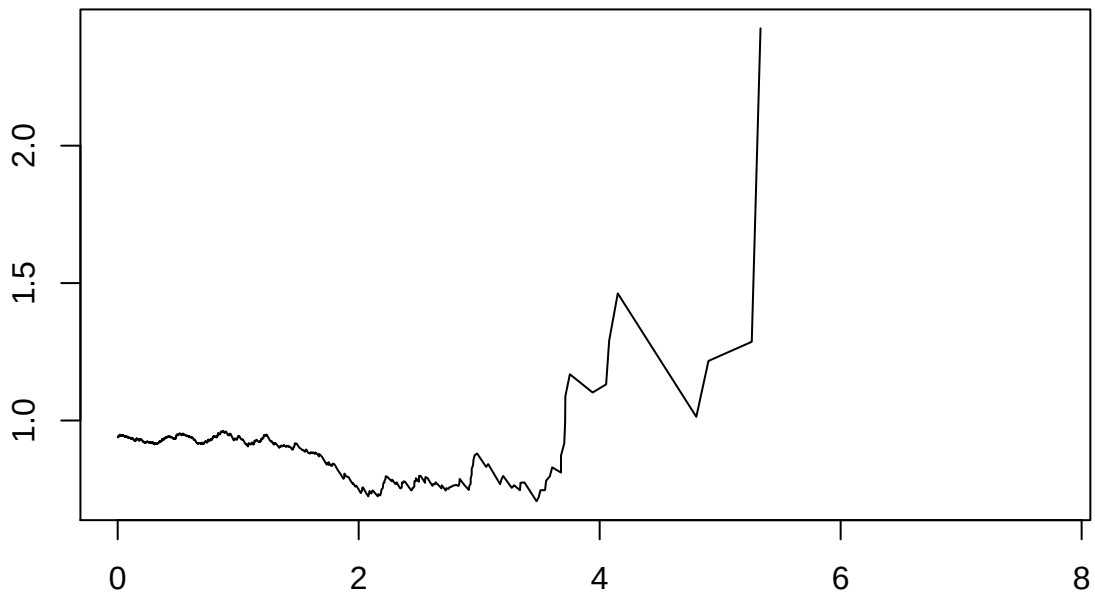

```
###########loi Gumbel###########
Nidd_gumbel=rgumbel(1000,0,1)
Nidd_gumbel_dec=sort(Nidd_gumbel, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_gumbel, decreasing = FALSE)
#Exces(Nidd_gumbel_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_gumbel_dec,seuil_stat_ord),
type='l',main='Gumbel',xlab = "" , ylab = "")
```



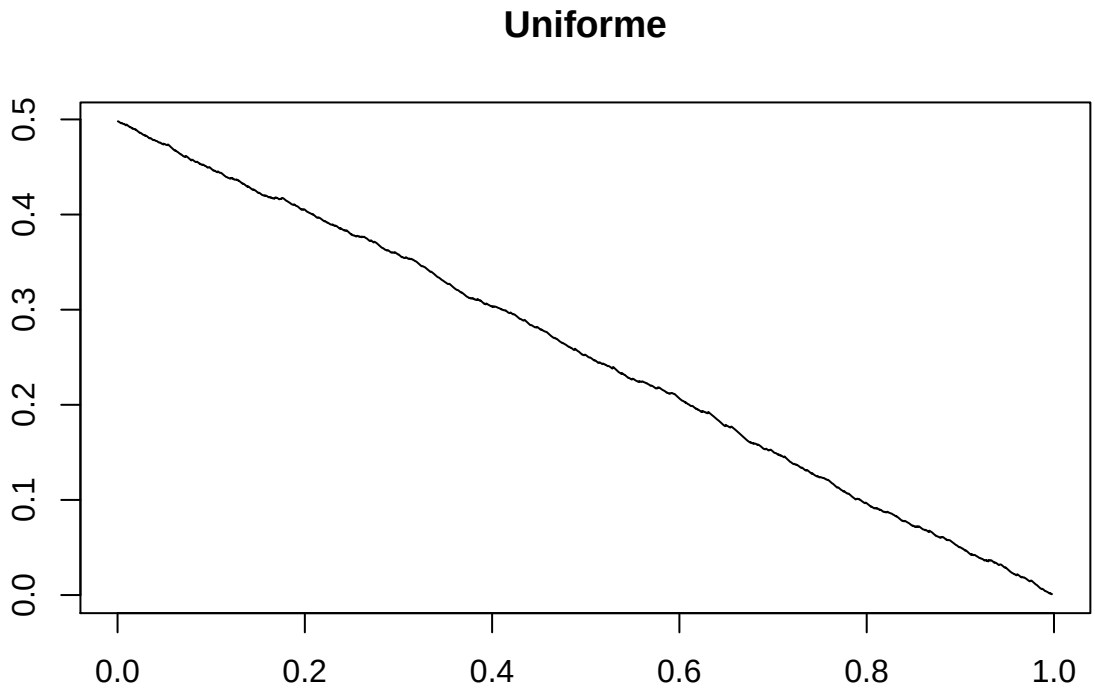

```
############loi Frechet############
Nidd_frechet=rfrechet(1000, loc=0, scale=1, shape=1)
Nidd_frechet_dec=sort(Nidd_frechet, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_frechet, decreasing = FALSE)
#Exces(Nidd_frechet_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_frechet_dec,seuil_stat_ord),
type='l',main='Frechet',xlab = "" , ylab = "")
```



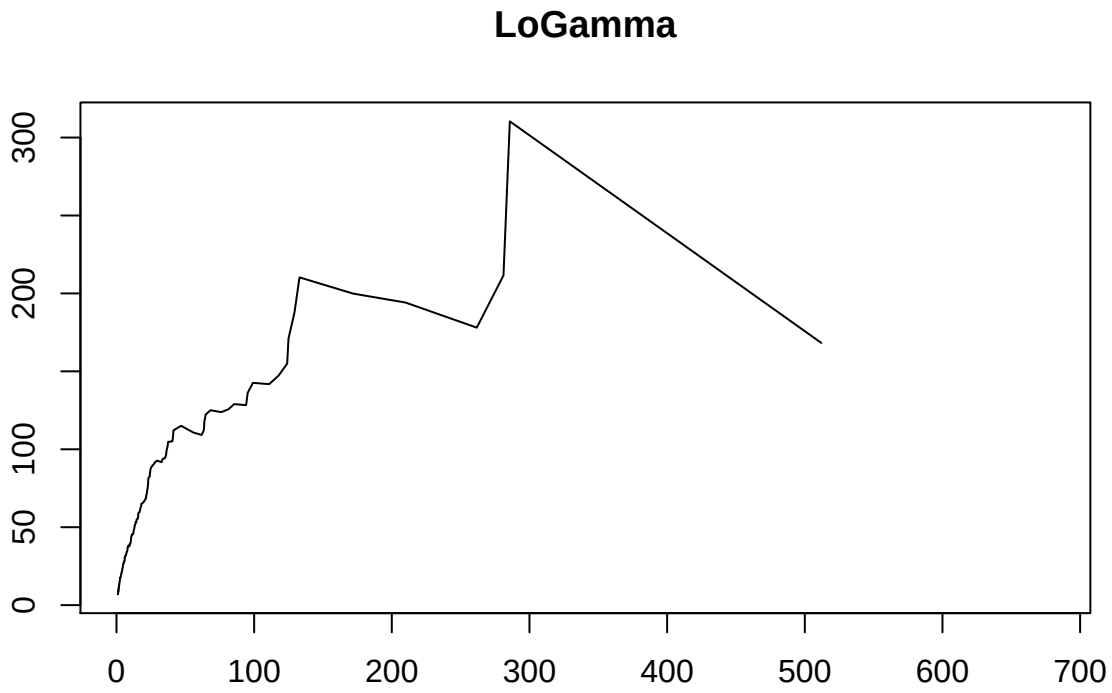

```
############loi Exponentielle############
Nidd_Expo=rexp(1000,1)
Nidd_Expo_dec=sort(Nidd_Expo, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_Expo, decreasing = FALSE)
#Exces(Nidd_Expo_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_Expo_dec,seuil_stat_ord),
type='l',main='Expo',xlab = "" , ylab = "")
```



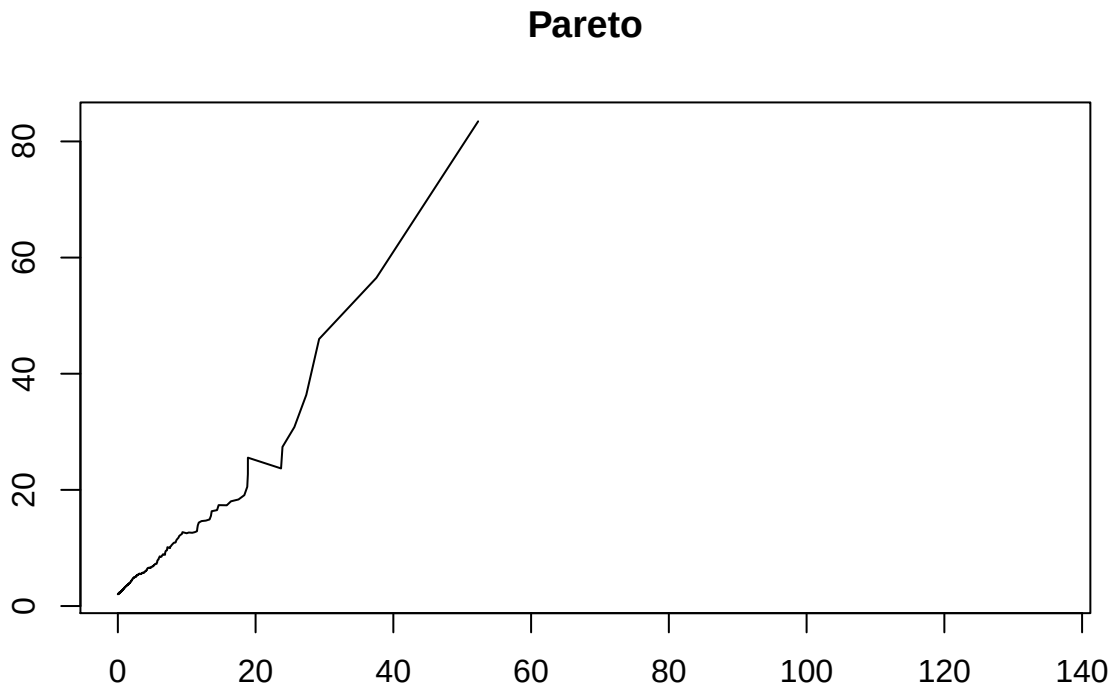

```
############loi Uniforme############
Nidd_unifo=runif(1000,0,1)
Nidd_unifo_dec=sort(Nidd_unifo, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_unifo, decreasing = FALSE)
#Exces(Nidd_unifo_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_unifo_dec,seuil_stat_ord),
type='l',main='Uniforme',xlab = "" , ylab = "")
```




```
###########loi Gamma###########
Nidd_LoGamma=rlgamma(1000,1,1)
Nidd_LoGamma_dec=sort(Nidd_LoGamma, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_LoGamma, decreasing = FALSE)
#Exces(Nidd_Gamma_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_LoGamma_dec,seuil_stat_ord),
type='l',main='LoGamma',xlab = "" , ylab = "")
```




```
###########loi Pareto###########
Nidd_Pareto=rpareto(1000,2,2)
Nidd_Pareto_dec=sort(Nidd_Pareto, decreasing = TRUE)
seuil_stat_ord=sort(Nidd_Pareto, decreasing = FALSE)
#Exces(Nidd_Pareto_dec,seuil_stat_ord)
#par(mfrow=c(2,2))
#plot(sort(nidd2),Exces(nidd2,sort(nidd2)),type='l')
plot(seuil_stat_ord,Exces(Nidd_Pareto_dec,seuil_stat_ord),
type='l',main='Pareto',xlab = "" , ylab = "")
```



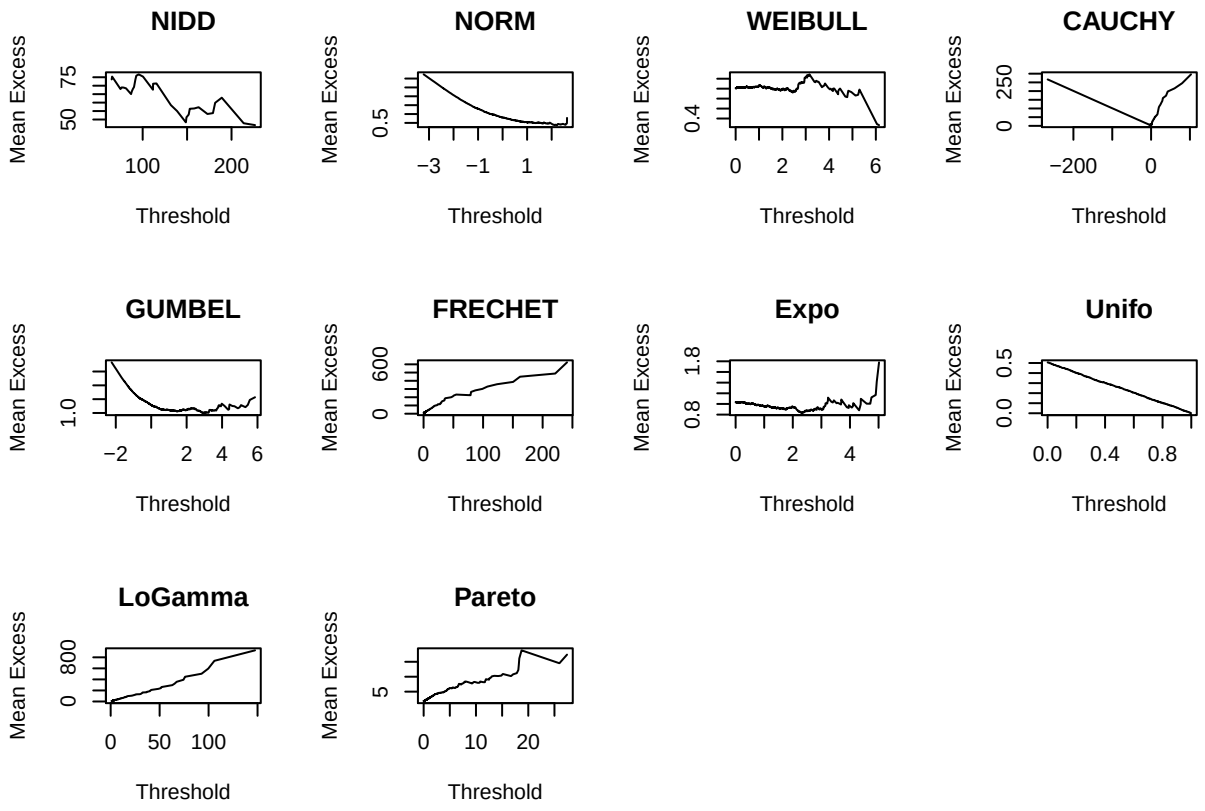
- utilisons la fonction ‘”*meplot*” du package ”*evir*” pour tracer la fonction empirique des excès moyens pour différentes lois.

```
Normal = rnorm(1000,0,1)
Weibull = rweibull(1000, 1)
Cauchy = rcauchy(1000, 0, 1)
Gumbel = rgumbel(1000,0,1)
Frechet = rfrechet(1000, loc=0, scale=1, shape=1)
Expo=rexp(1000,1)
Unifo=runif(1000,0,1)
LoGamma=rlgamma(1000,1,1)
Pareto=rpareto(1000,2,2)


close.screen( all = TRUE )
```

```
## [1] FALSE
```

```
par(mfrow=c(3,4))
meplot(nidd, 3, type="l", main="NIDD")
meplot(Normal, 3, type="l", main="NORM")
meplot(Weibull, 3, type="l", main="WEIBULL")
meplot(Cauchy, 3, type="l", main="CAUCHY")
meplot(Gumbel, 3, type="l", main="GUMBEL")
meplot(Frechet, 3, type="l", main="FRECHET")
meplot(Expo, 3, type="l", main="Expo")
meplot(Unifo, 3, type="l", main="Unifo")
```

```r
meplot(LoGamma, 3, type="l", main="LoGamma")
meplot(Pareto, 3, type="l", main="Pareto")
```



Application 2 : On termine le seuil avec le Graphique de $HILL$

Le graphique de $Hill$ est un estimateur qui est défini uniquement pour $\psi > 0$, c'est-à-dire dans le cas où la distribution des valeurs extrêmes correspond à une distribution de Fréchet. Le seuil, dans ce cas, correspond à la valeur à partir de laquelle la fonction de $Hill$ est assimilable à une droite horizontale.

Implémentation

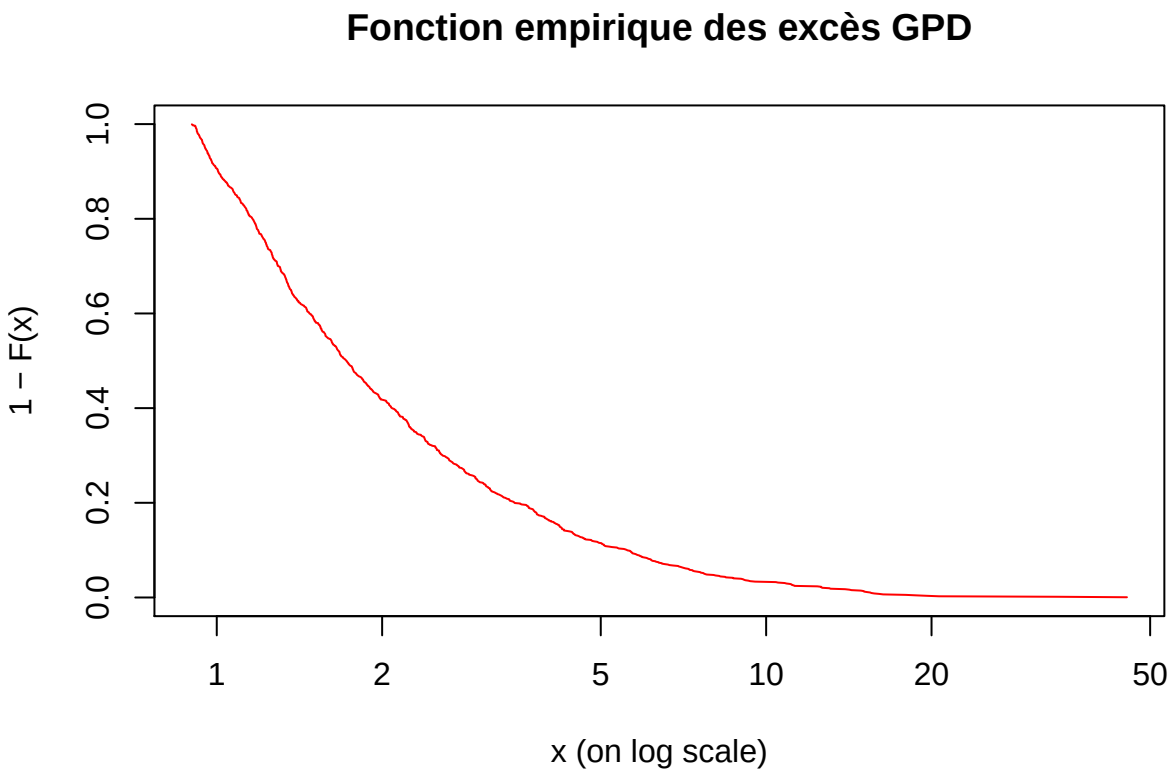
1. Commençons par simuler une GPD à l'aide de $rgpd$:

```r
GP=rgpd(1000, 0.4, 0.9)
#GP=rgpd(1000, 0.30)
#GP=rgpd(500,0.4,0,1)
head(GP, n=3)
```

```
## [1] 1.6468944 1.8121666 0.9283203
```

2. On trace la fonction de survie empirique des excès à l'aide de la fonction $emplot$:

```r
emplot(GP,type="l",main="Fonction empirique des excès GPD",
col="red")
```



3. On utilise la fonction *gpd* pour calculer les estimateurs de γ et σ:

Par la méthode de maximum de vraisemblance

```
Estim_ml=gpd(GP,0,method = "ml")$par.ests
Estim_pwm=gpd(GP,0,method = "pwm")$par.ests
Estim_ml
```

```
##         xi       beta
## 0.04691439 2.62299295
```

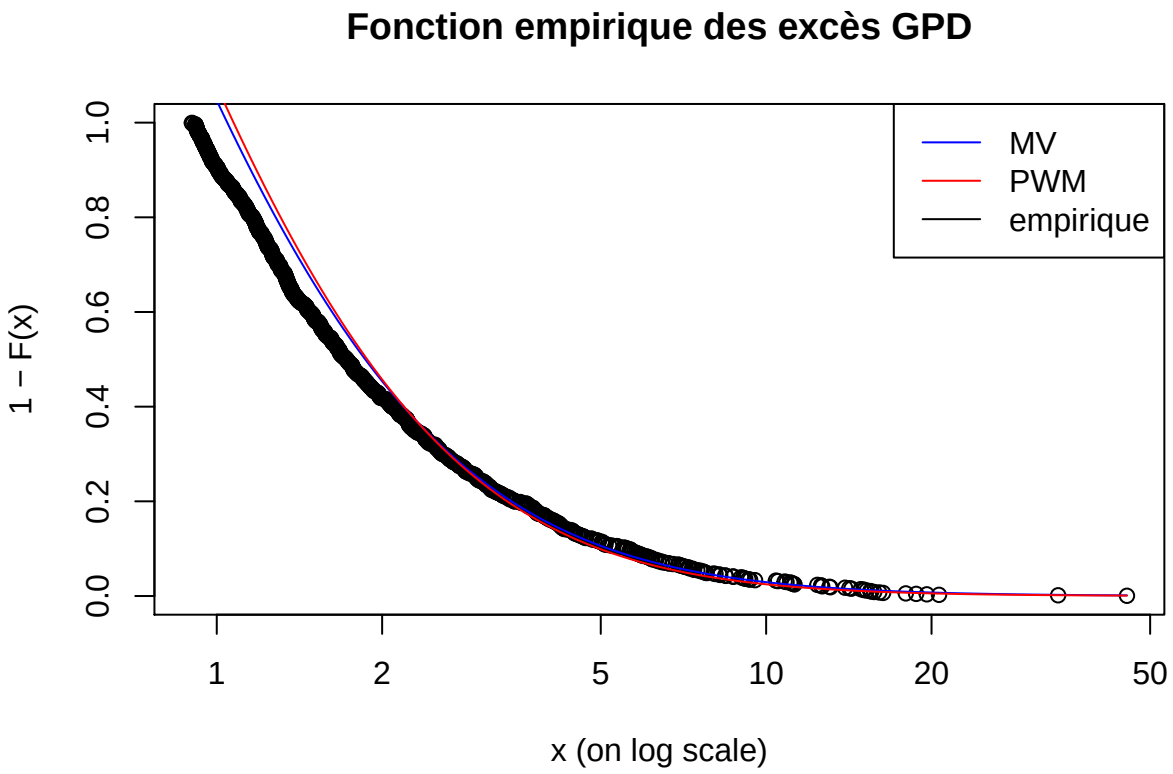
```
Estim_pwm
```

```
##         xi       beta
## -0.3410325  3.6976814
```

4. On trace les fonctions de survie des deux GPD estimés sur un même graphe:

```
fgpd=function(loi,x){
    n=length(loi)

    sigma_ml=gpd(loi, nextremes=n, method="ml")$par.ests[1]
    gamma_ml=gpd(loi, nextremes=n, method="ml")$par.ests[2]
    sigma_pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[1]
    gamma_pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[2]

    emplot(GP, main="Fonction empirique des excès GPD")
    lines(x,1-pgpd(x,sigma_ml,gamma_ml),col="blue",
    main="Fonction
```

```
    de survie",type="l")
    lines(x,1-pgpd(x,sigma_pwm,gamma_pwm),col="red")
    legend(legend=c("MV","PWM","empirique"),"topright",
    col=c("blue","red","black"),lty=c(1,1,1))
}
x=sort(GP)
x=seq(x[1],x[length(x)], 0.01)
fgpd(GP,x)
```

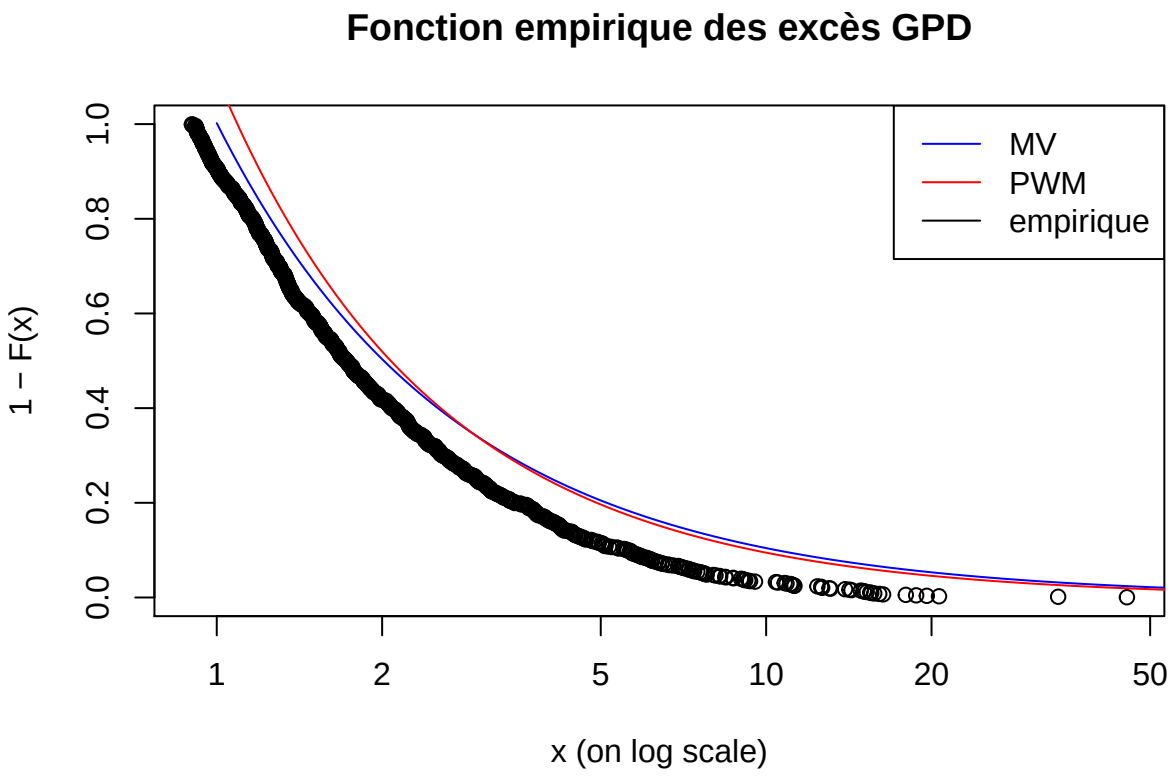


5. Appliquons ceci à d'autres lois usuelles

Loi LogGamma

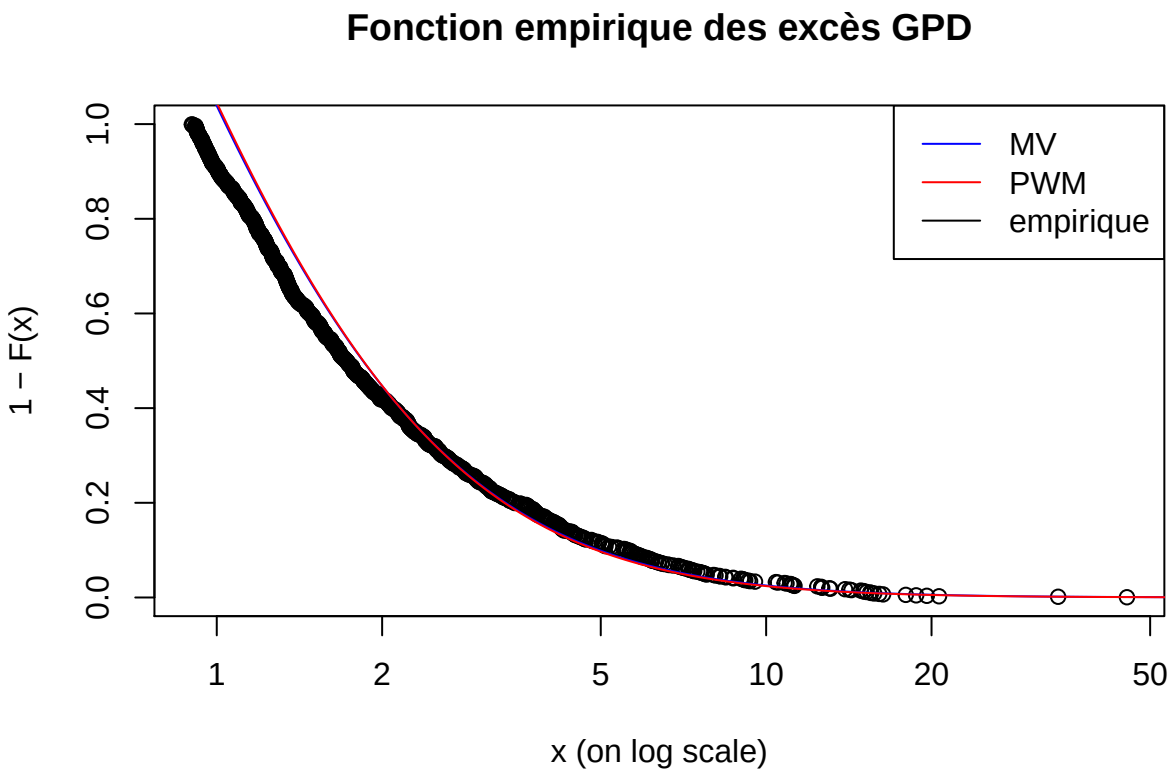
```
logamma=rlgamma(1000,1,1)
x=sort(logamma)
x=seq(x[1],x[length(x)], 0.01)
fgpd(logamma,x)
```

```
## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5

## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5
```



Loi Pareto

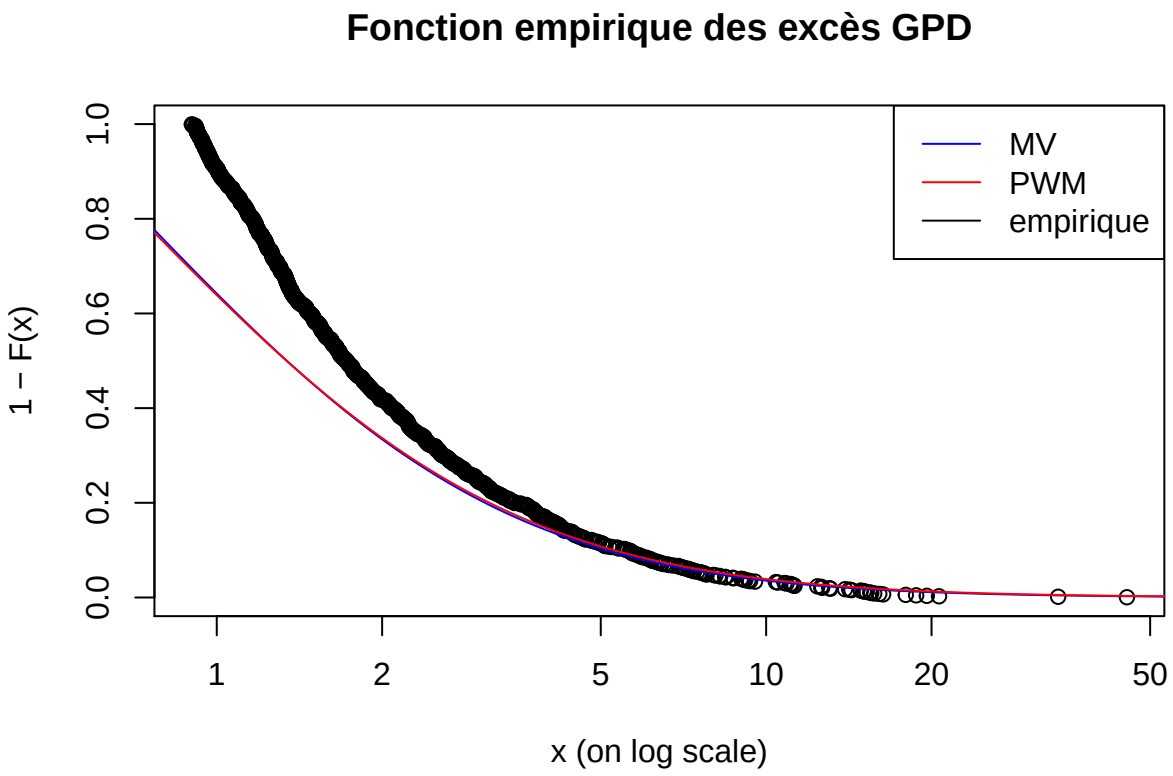
```
Pareto=rpareto(1000,2,2)
x=sort(Pareto)
x=seq(x[1],x[length(x)], 0.01)
fgpd(Pareto,x)
```



Loi Burr

```
bur=rburr(1000,1,1,2)
x=sort(bur)
x=seq(x[1],x[length(x)], 0.01)
fgpd(bur,x)
```

```
## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5

## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5
```

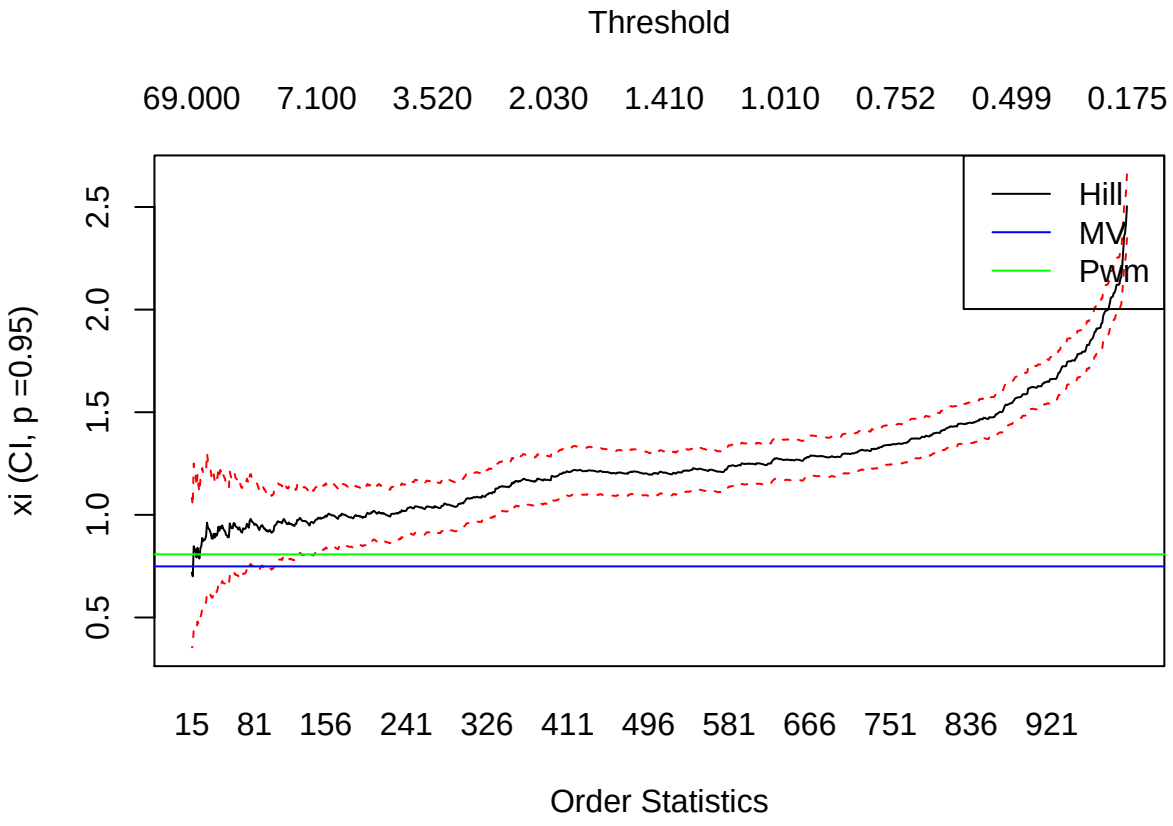


6. On trace sur un même graphique les 3 estimateurs γ: PWM, EMV, $Hill$

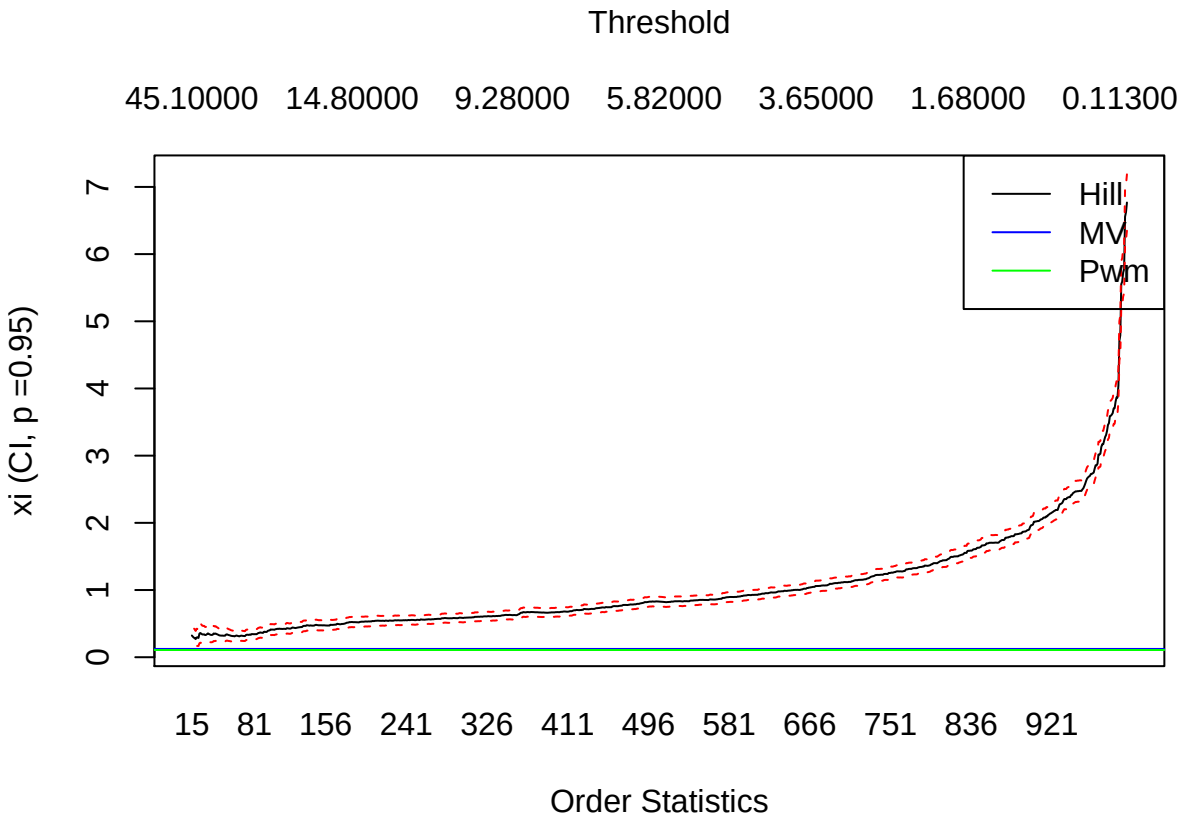
```
f_hill=function(loi,n){
  ml=gpd(loi, nextremes=n, method="ml")$par.ests[1]
  pwm=gpd(loi, nextremes=n, method="pwm")$par.ests[1]
  #plot(ml)
  hill(loi, "xi", xlim=c(15,n))
  abline(h=ml, col="blue")
  abline(h=pwm, col="green")
  #dev.off()
  legend(legend=c("Hill", "MV", "Pwm"),"topright",  col=c("black","blue","green"), lty=c(1,1,1))
  #legend(legend=c("MV","PWM","empirique"),"topright",col=c("red","blue","black"),lty=c(1,1,1))

}
```

```
#######Loi de Frechet #######
frechet=rfrechet(1000, loc=0, scale=1, shape=1)
f_hill(frechet,1000)
```

```
## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5
```

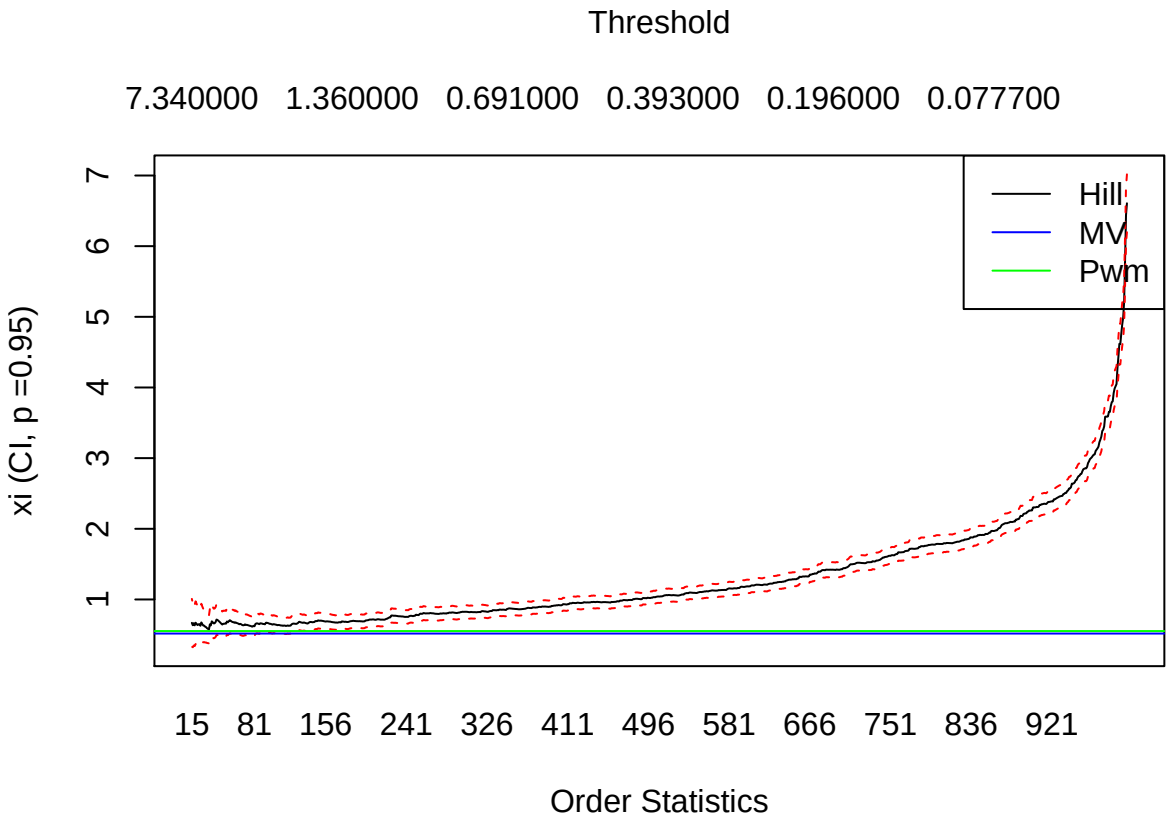



```
#######Loi Pareto #######
Pareto=rpareto(1000,10,15)
f_hill(Pareto,1000)
```

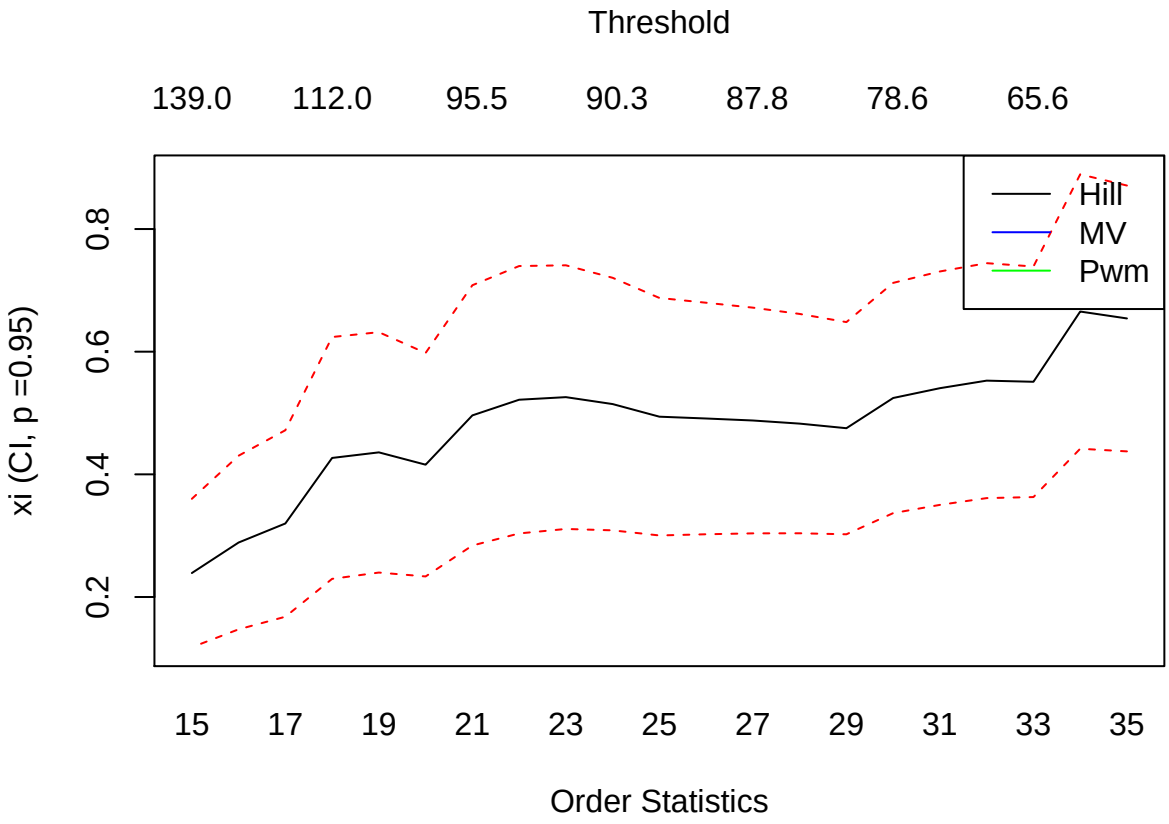



```
####### loi de Burr #######
bur=rburr(1000,1,1,2)
f_hill(bur,1000)
```

```
## Warning in gpd(loi, nextremes = n, method = "pwm"): Asymptotic standard errors
## not available forPWM Method when xi > 0.5
```




```
f_hill(nidd,length(nidd))
```



En conclusion, à partir des graphes obtenus précédemment, pour les lois considérées,la fonction $Hill$ devient une droite horizontale aux environs de la valeur 260, est l'estimateur de $Hill$ est proche de EMV et PWM.

On peut donc considérer que le domaine d'attraction de la distribution des crues est $Frechet$.

3. Package graphics R

3.1. Introduction

Le logiciel R est l'un des logicielsincontestablement reconnu pour ses graphiques tels que Les bandes linéaires les bandes en pointillé ou même les bandes de nuages, ...

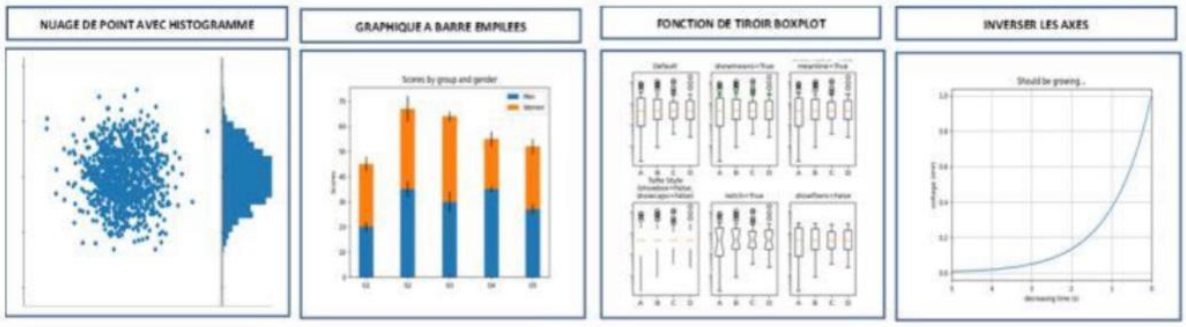


Figure 1: Quelques exemples de graphiques

R a intégré des bibliothèques qui offrent un excellent support graphique. L'installation R contient trois packages importants :

1. les graphiques
2. les treillis
3. la grille

Ces packages fournissent des outils pour dessiner une grande variété de tracés et de formes. De plus, de nombreux packages externes tels que `ggplot2` peuvent être téléchargés dans R.

QU'EST-CE QUE LE BUT D'UN GRAPHIQUE

- Mieux comprendre le problème
- Mieux expliquer un phénomène
- Le but est de nous nous aider à rendre une situation abstraite en information convaincante.

Pour faire ces graphiques , le logiciels "R" a mis à disposition un package à cet effet qui , de par ces multiples commandes met en exécution une fonction de tracé.

Nous allons donner une brève description de ces trois packages en mettant un accent particulier sur la commande "$PLOT$" ci-dessous.

3.2. Description et installation d'un package graphics R

La bibliothèque graphique est le package graphique de base standard fourni avec l'installation R. Cette bibliothèque a été développée à l'origine pour R. De nombreux tracés standards tels que des tracés de base de points et de lignes, des histogrammes, des graphiques à barres, des camemberts, des bplots, etc. peuvent être dessinés avec cette bibliothèque.

À toutes fins pratiques, la bibliothèque graphique est suffisamment silencieuse pour créer des tracés et des graphiques d'excellente qualité que nous utilisons généralement pour l'analyse des données et des statistiques.

Ce package est fourni avec l'installation de R.

Le paquet de grille

Le système graphique de grille a ensuite été développé par Paul Murrel et ajouté à R. Il s'agit d'un système graphique de bas niveau qui permet de dessiner et d'organiser des formes géométriques de base telles que des

polygones, des courbes, des images raster, etc. Le paquet de grille contient des fonctions pour accéder au canevas et permet la création de plusieurs régions appelées fenêtres sur une seule toile.

Ce package est fourni avec l'installation *R*. Nous devons charger la bibliothèque de grille dans *R* avant de l'utiliser. (Le package graphique se charge automatiquement lorsque nous démarrons *R*).

Pour charger la bibliothèque de grille dans *R*, tapez `grille`

Le paquet lattice

Le treillis , développé par Deepayan Sarkar, est un système de visualisation de haut niveau basé sur une méthode appelée graphique en treillis . Ce package gère très efficacement les données multivariées.

Le package lattice se compose de fonctions de haut niveau pour chaque tâche. Ces fonctions renvoient des objets qui peuvent être convertis en graphiques par les fonctions `plot()` du package *R* de base. Ce package est basé sur le moteur graphique de grille mentionné ci-dessus.

Ce package est également fourni avec l'installation de *R* et nécessite le chargement du package `grDevices`.

Pour charger la bibliothèque grDevices dans *R*, tapez

`grDevices`

Le paquet ggplot2

`ggplot2` est une bibliothèque graphique pour *R*, créée par Hadley Wiskham. Il est mentionné dans sa page d'accueil que `ggplot2` est un système de traçage pour *R*, basé sur la grammaire des graphiques, qui essaie de prendre les bonnes parties des graphiques de base et de treillis et aucune des mauvaises parties". (Le Grammer of Graphics mentionné ici est un livre classique sur les méthodes graphiques pour la

visualisation de données scientifiques écrit par Leyland Wilkinson). Nous pouvons créer des parcelles très élégantes avec cette bibliothèque.

`ggplot2` est un package externe qui doit être téléchargé de l'intérieur *R*. Pour installer ce package en ligne à partir de l'invite *R*, tapez: `ggplot2`

3.3. Applications

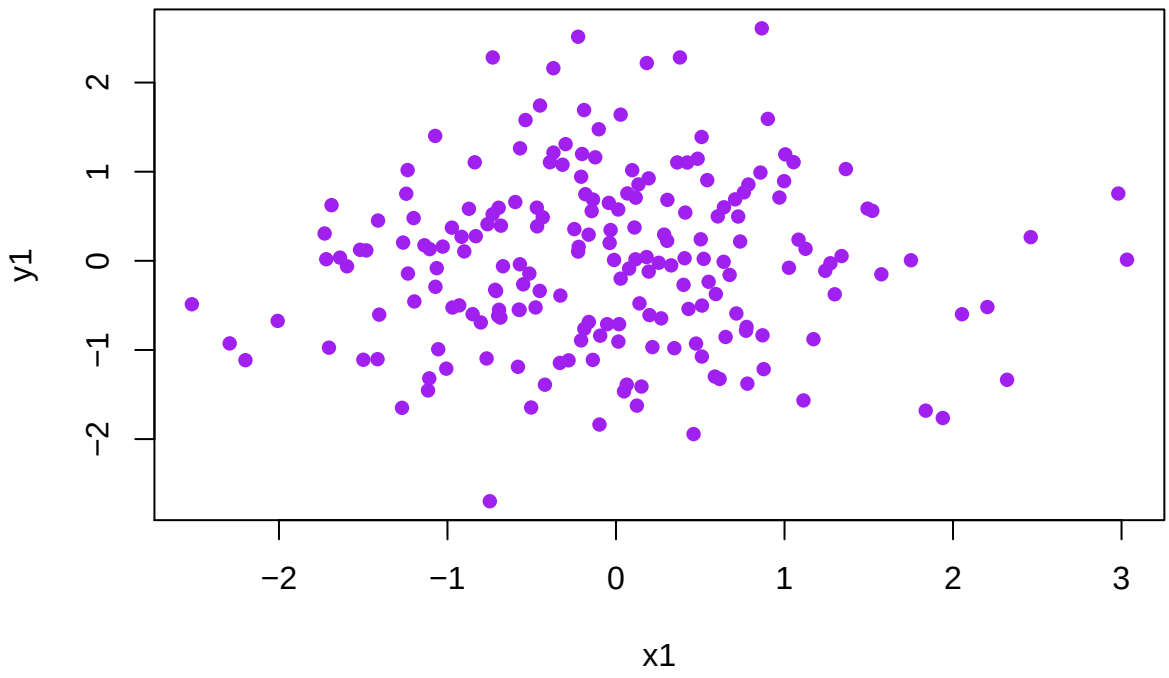
R avec `plot()`, `points()`, `lines()`, `polygon`

La fonction `plot()` **black** trace les points et les lignes.

Les tracés de points et de lignes peuvent être produits en utilisant la fonction `terrain()`, qui prend les points x et y sous forme de vecteurs ou de nombre unique avec de nombreux autres paramètres. Les paramètres x et y sont nécessaires. Pour d'autres, la valeur par défaut sera utilisée en l'absence de la valeur. Dans les lignes de commande ci-dessous, nous créons d'abord une paire de séquences x et y et les passons en paramètres à la fonction `terrain()`.

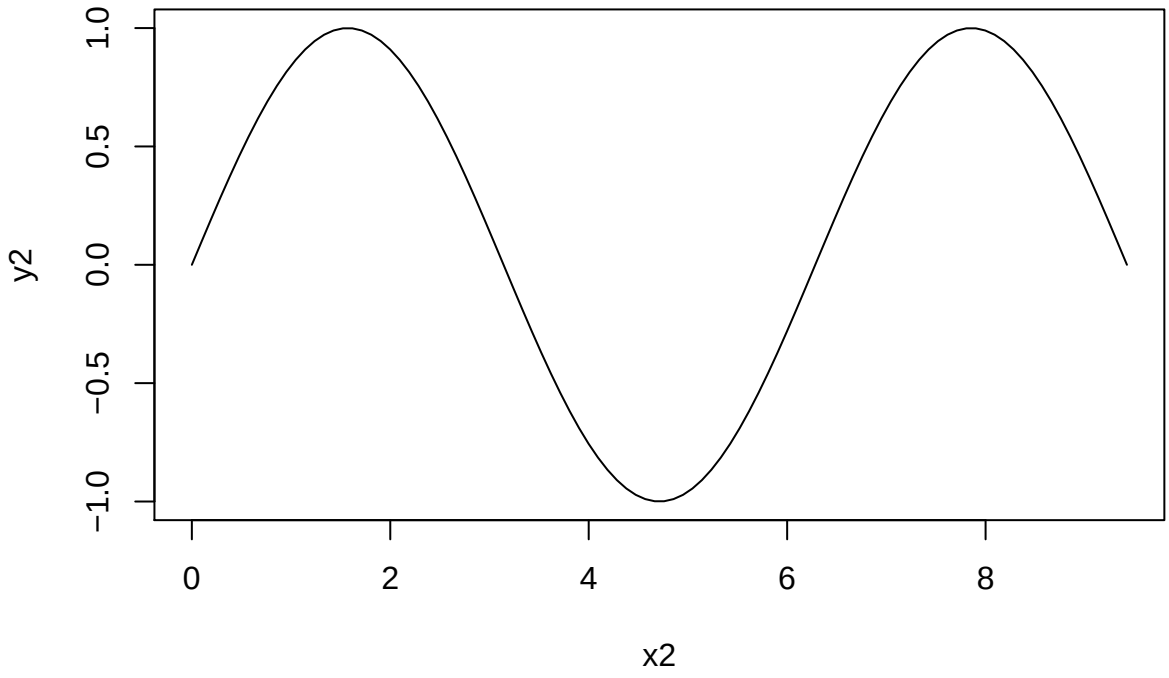
Commande d'exécution de plot avec un graphique en nuage de point

```
#Comment représenter graphiquement la commande plot avec des nuages de #points
x1<-rnorm(200,mean = 0, sd=1)
y1<-rnorm(200,0,1)
plot(x1,y1,pch=16,col="purple")
```

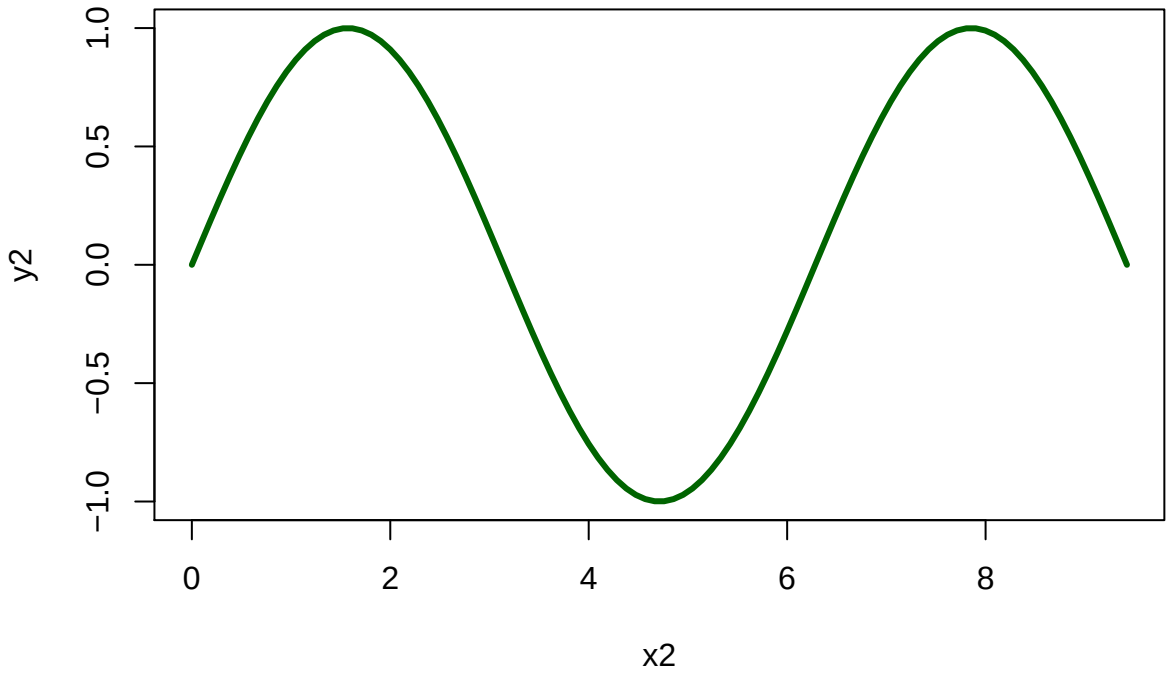


Commande d'exécution de plot avec un graphique en ligne

```
x2<-seq(0,3*pi,len=100)
y2<-sin(x2)
plot(x2,y2,type ="l")
```

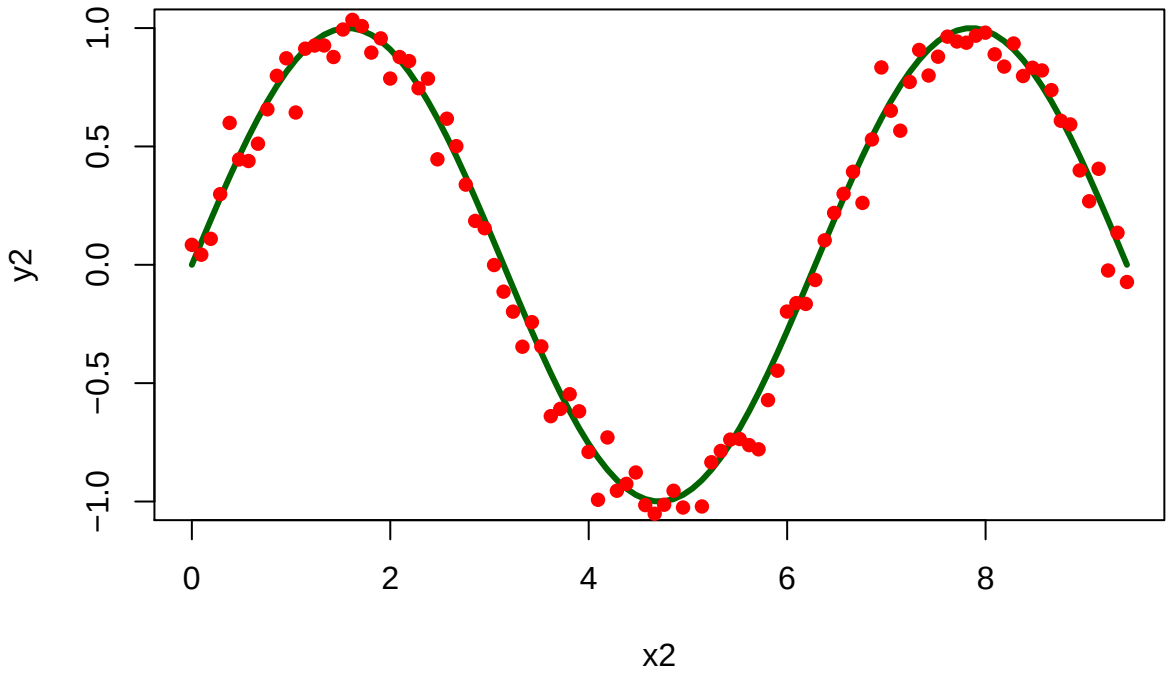



```
plot(x2,y2,type ="l",lwd=3,col="darkgreen")
```



Comment représenter graphiquement la commande plot avec des nuages de points

```
x2<-seq(0,3*pi,len=100)
y2<-sin(x2)
plot(x2,y2,type ="l",lwd=3,col="darkgreen")
y2.rand<-y2+rnorm(100,0,0.1)
points(x2,y2.rand,pch=16,col="red")
```


y2
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
x2